DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.

Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve:

- How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
- How to increase the consistency of project vetting across different volunteers to improve the experience for teachers
- How to focus volunteer time on the applications that need the most assistance

The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.

About the DonorsChoose Data Set

The `train.csv` data set provided by DonorsChoose contains the following features:

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the follo • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **E** • `Literacy` • `Literature & Writing, Social Sciences`

Feature	Description
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to mana`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*
`project_essay_4`	Fourth application essay*
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-2`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teache

* See the section **Notes on the Essay Data** for more details about these features.

Additionally, the `resources.csv` data set provides more data about the resources required for each project. Each line in this file represents a resource required by a project:

Feature	Description
`id`	A `project_id` value from the `train.csv` file. **Example:** `p036502`
`description`	Desciption of the resource. **Example:** `Tenor Saxophone Reeds, Box of 25`
`quantity`	Quantity of the resource required. **Example:** `3`
`price`	Price of the resource required. **Example:** `9.95`

Note: Many projects require multiple resources. The `id` value corresponds to a `project_id` in train.csv, so you use it as a key to retrieve all resources needed for a project:

The data set contains the following label (the value you will attempt to predict):

Label	Description
`project_is_approved`	A binary flag indicating whether DonorsChoose approved the project. A value of `0` indicates the project \

Notes on the Essay Data

Prior to May 17, 2016, the prompts for the essays were as follows:

- **project_essay_1:** "Introduce us to your classroom"
- **project_essay_2:** "Tell us more about your students"
- **project_essay_3:** "Describe how your students will use the materials you're requesting"
- **project_essay_3:** "Close by sharing why your project will make a difference"

Starting on May 17, 2016, the number of essays was reduced from 4 to 2, and the prompts for the first 2 essays were changed to the following:

- **project_essay_1:** "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- **project_essay_2:** "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

For all projects with project_submitted_datetime of 2016-05-17 and later, the values of project_essay_3 and project_essay_4 will be NaN.

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

```
/usr/local/lib/python3.6/dist-packages/smart_open/ssh.py:34: UserWarning: paramiko
  warnings.warn('paramiko missing, opening SSH/SCP/SFTP paths will be disabled.  `p
```

```
import pickle
# Load the Drive helper and mount
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=94

Enter your authorization code:
··········
Mounted at /content/drive
```

```
dir_path = '/content/drive/My Drive/appliedai/Data'
!ls /content/drive/My\ Drive/appliedai
```

```
Data  Dumps  KNN  New_dumps  tsne
```

1.1 Reading Data

```
project_data = pd.read_csv(os.path.join(dir_path,'train_data.csv'))
resource_data = pd.read_csv(os.path.join(dir_path,'resources.csv'))
print(project_data.shape)
print(resource_data.shape)
```

```
(109248, 17)
(1541272, 4)
```

```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (109248, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_s
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.column


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)


# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
project_data = project_data[cols]


project_data.head(2)
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school_
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	

```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

```
Number of data points in train data (1541272, 4)
['id' 'description' 'quantity' 'price']
```

1.2 preprocessing of `project_subject_categories`

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth",
        if 'The' in j.split(): # this will split each of the catogory based on space "Math
            j=j.replace('The','') # if we have the words "The" we are going to replace it
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)

from collections import Counter
my_counter = Counter()
for word in project_data['clean_categories'].values:
    my_counter.update(word.split())

cat_dict = dict(my_counter)
sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
```

1.3 preprocessing of `project_subject_subcategories`

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth",
        if 'The' in j.split(): # this will split each of the catogory based on space "Math
            j=j.replace('The','') # if we have the words "The" we are going to replace it
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))
```

1.3 Text preprocessing

1.3.1 Essay Text

```
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

```
project_data.head(2)
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school_
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	

```
type(project_data['project_is_approved'][0])
```

```
numpy.int64
```

```
# printing some random essays.
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
print(project_data['essay'].values[99999])
print("="*50)
```

```
I have been fortunate enough to use the Fairy Tale STEM kits in my classroom as wel
==================================================
I teach high school English to students with learning and behavioral disabilities.
==================================================
```

```python
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

```python
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
\"A person is a person, no matter how small.\" (Dr.Seuss) I teach the smallest stud
==================================================
```

```python
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-pytho
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
 A person is a person, no matter how small.  (Dr.Seuss) I teach the smallest studen
```

```python
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
 A person is a person no matter how small Dr Seuss I teach the smallest students wi
```

```python
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're",
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'hi
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', '
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', '
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over'
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', '
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'd
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn'
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn
            'won', "won't", 'wouldn', "wouldn't"]
```

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████| 109248/109248 [01:04<00:00, 1683.01it/s]
```

```
# after preprocesing
preprocessed_essays[20000]
cleaned_project_data = pd.DataFrame()
# project_data['cleaned_essay'] = preprocessed_essays
cleaned_project_data['id'] = project_data['id']
cleaned_project_data['cleaned_essay'] = preprocessed_essays
cleaned_project_data.head(2)
```

	id	cleaned_essay
55660	p205479	i fortunate enough use fairy tale stem kits cl...
76127	p043609	imagine 8 9 years old you third grade classroo...

1.3.2 Project title Text

```
# printing some random essays.
print(project_data['project_title'].values[0])
print("="*50)
print(project_data['project_title'].values[150])
print("="*50)
print(project_data['project_title'].values[1000])
print("="*50)
print(project_data['project_title'].values[20000])
print("="*50)
print(project_data['project_title'].values[99999])
print("="*50)
```

```
Engineering STEAM into the Primary Classroom
==================================================
Building Blocks for Learning
==================================================
Empowering Students Through Art:Learning About Then and Now
==================================================
Health Nutritional Cooking in Kindergarten
==================================================
Turning to Flexible Seating: One Sixth-Grade Class's Journey to Freedom
==================================================
```

```
# similarly you can preprocess the titles also
preprocessed_titles = []
for sentance in tqdm(project_data['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_titles.append(sent.lower().strip())
```

```
100%|██████████| 109248/109248 [00:03<00:00, 35225.26it/s]
```

```
# project_data['cleaned_project_title'] = preprocessed_titles
cleaned_project_data['cleaned_project_title'] = preprocessed_titles
cleaned_project_data['clean_categories'] = project_data['clean_categories']
cleaned_project_data['clean_subcategories'] = project_data['clean_subcategories']
cleaned_project_data['project_is_approved'] = project_data['project_is_approved']
cleaned_project_data['teacher_prefix'] = project_data['teacher_prefix']
cleaned_project_data['school_state'] = project_data['school_state']
cleaned_project_data['project_grade_category'] = project_data['project_grade_category']
cleaned_project_data['teacher_number_of_previously_posted_projects'] = project_data['teach
# cleaned_project_data.head(5)
cleaned_project_data.to_csv(os.path.join(dir_path,'cleaned_project_data.csv'))
```

```
cleaned_project_data.head(2)
```

	id	cleaned_essay	cleaned_project_title	clean_categories	clean_subca
55660	p205479	i fortunate enough use fairy tale stem kits cl...	engineering steam primary classroom	Math_Science	Applie Health_L
76127	p043609	imagine 8 9 years old you third grade classroo...	sensory tools focus	SpecialNeeds	Spe

```
cleaned_project_data['cleaned_text'] = cleaned_project_data['cleaned_essay'].map(str) + \
                                       cleaned_project_data['cleaned_project_title'].map(
cleaned_project_data.head(2)
```

	id	cleaned_essay	cleaned_project_title	clean_categories	clean_subca
55660	p205479	i fortunate enough use fairy tale stem kits cl...	engineering steam primary classroom	Math_Science	Applie Health_L
76127	p043609	imagine 8 9 years old you third grade classroo...	sensory tools focus	SpecialNeeds	Spe

Assignment 3: Apply KNN

1. **[Task-1] Apply KNN(brute force version) on these feature sets**

- Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
- Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
- Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
- Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

2. **Hyper paramter tuning to find best K**
    - Find the best hyper parameter which results in the maximum AUC value
    - Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
    - Use gridsearch-cv or randomsearch-cv or write your own for loops to do this task

3. **Representation of results**
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure
    - Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.
    - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

4. **[Task-2]**
    - Select top 2000 features from feature Set 2 using `SelectKBest` and then apply KNN on top of these features

    -
      ```
                      from sklearn.datasets import load_digits
                      from sklearn.feature_selection import
      SelectKBest, chi2
                      X, y = load_digits(return_X_y=True)
                      X.shape
                      X_new = SelectKBest(chi2, k=20).fit_transform(X,
      y)
                      X_new.shape
                      ========
                      output:
                      (1797, 64)
                      (1797, 20)
      ```
    - Repeat the steps 2 and 3 on the data matrix after feature selection

5. **Conclusion**
    - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

2. K Nearest Neighbor

2.1 Splitting data into Train and cross validation(or test): Stratified Sampling

```
project_65k = cleaned_project_data[:65000].copy()
project_65k.shape
```

```
(65000, 11)
```

```
X = project_65k.drop(['project_is_approved'], axis=1)
y = project_65k['project_is_approved']
print(type(y))
```

```
<class 'pandas.core.series.Series'>
```

```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify
```

```
with open("/content/drive/My Drive/appliedai/New_dumps/y_train.pkl","wb") as file:
  pickle.dump(y_train, file)

with open("/content/drive/My Drive/appliedai/New_dumps/y_cv.pkl","wb") as file:
  pickle.dump(y_cv, file)

with open("/content/drive/My Drive/appliedai/New_dumps/y_test.pkl","wb") as file:
  pickle.dump(y_test, file)
```

```
print(len(X_train), len(y_train))
print(len(X_cv), len(y_cv))
print(len(X_test), len(y_test))
```

```
29178 29178
14372 14372
21450 21450
```

```
print(set(X_train['teacher_prefix'].values))
print(set(X_cv['teacher_prefix'].values))
print(set(X_test['teacher_prefix'].values))
```

```
{nan, 'Mr.', 'Ms.', 'Teacher', 'Mrs.', 'Dr.'}
{'Mr.', 'Ms.', 'Teacher', 'Mrs.', 'Dr.'}
{nan, 'Mr.', 'Ms.', 'Teacher', 'Mrs.', 'Dr.'}
```

2.2 Make Data Model Ready: encoding numerical, categorical features

Project Category

```
# we use count vectorizer to convert the values into one hot encoded features
# Project Category
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, bin
X_train_one_hot_clean_cat = vectorizer.fit_transform(X_train['clean_categories'].values)
X_cv_one_hot_clean_cat = vectorizer.transform(X_cv['clean_categories'].values)
X_test_one_hot_clean_cat = vectorizer.transform(X_test['clean_categories'].values)
print(vectorizer.get_feature_names())

print("Shape of matrix after one hot encodig ",X_train_one_hot_clean_cat.shape)
print("Shape of matrix after one hot encodig ",X_cv_one_hot_clean_cat.shape)
print("Shape of matrix after one hot encodig ",X_test_one_hot_clean_cat.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'Speci
Shape of matrix after one hot encodig  (29178, 9)
Shape of matrix after one hot encodig  (14372, 9)
Shape of matrix after one hot encodig  (21450, 9)
```

```
with open("/content/drive/My Drive/appliedai/New_dumps/X_train_one_hot_clean_cat.pkl","wb"
  pickle.dump(X_train_one_hot_clean_cat, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_cv_one_hot_clean_cat.pkl","wb") a
  pickle.dump(X_cv_one_hot_clean_cat, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_test_one_hot_clean_cat.pkl","wb")
  pickle.dump(X_test_one_hot_clean_cat, file)
```

Project Sub-category

```
# Project Sub-category
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False,
X_train_one_hot_clean_sub_cat = vectorizer.fit_transform(X_train['clean_subcategories'].va
X_cv_one_hot_clean_sub_cat = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_one_hot_clean_sub_cat = vectorizer.transform(X_test['clean_subcategories'].values)
print(vectorizer.get_feature_names())

print("Shape of matrix after one hot encodig ",X_train_one_hot_clean_sub_cat.shape)
print("Shape of matrix after one hot encodig ",X_cv_one_hot_clean_sub_cat.shape)
print("Shape of matrix after one hot encodig ",X_test_one_hot_clean_sub_cat.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extrac
Shape of matrix after one hot encodig  (29178, 30)
Shape of matrix after one hot encodig  (14372, 30)
Shape of matrix after one hot encodig  (21450, 30)
```

```
with open("/content/drive/My Drive/appliedai/New_dumps/X_train_one_hot_clean_sub_cat.pkl",
  pickle.dump(X_train_one_hot_clean_sub_cat, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_cv_one_hot_clean_sub_cat.pkl","wb
  pickle.dump(X_cv_one_hot_clean_sub_cat, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_test_one_hot_clean_sub_cat.pkl","
  pickle.dump(X_test_one_hot_clean_sub_cat, file)
```

School State

```
# School State
vectorizer = CountVectorizer(lowercase=False, binary=True)
X_train_one_hot_clean_school_state = vectorizer.fit_transform(X_train['school_state'].valu
X_cv_one_hot_clean_school_state = vectorizer.transform(X_cv['school_state'].values)
X_test_one_hot_clean_school_state = vectorizer.transform(X_test['school_state'].values)
print(vectorizer.get_feature_names())

print("Shape of matrix after one hot encodig ",X_train_one_hot_clean_school_state.shape)
print("Shape of matrix after one hot encodig ",X_cv_one_hot_clean_school_state.shape)
print("Shape of matrix after one hot encodig ",X_test_one_hot_clean_school_state.shape)
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA', 'ID'
Shape of matrix after one hot encodig  (29178, 51)
Shape of matrix after one hot encodig  (14372, 51)
Shape of matrix after one hot encodig  (21450, 51)
```

```
with open("/content/drive/My Drive/appliedai/New_dumps/X_train_one_hot_clean_school_state.
  pickle.dump(X_train_one_hot_clean_school_state, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_cv_one_hot_clean_school_state.pkl
  pickle.dump(X_cv_one_hot_clean_school_state, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_test_one_hot_clean_school_state.p
  pickle.dump(X_test_one_hot_clean_school_state, file)
```

Teacher Prefix

```
# Teacher Prefix
import re
X_train_prefix_list = []
X_cv_prefix_list = []
X_test_prefix_list = []
for s in tqdm(X_train['teacher_prefix'].values):
    train_prefix = re.sub('[^A-Za-z0-9]+', '', str(s))
    train_prefix = re.sub('nan', '', str(train_prefix))
    X_train_prefix_list.append(train_prefix)
for s in tqdm(X_cv['teacher_prefix'].values):
    cv_prefix = re.sub('[^A-Za-z0-9]+', '', str(s))
    X_cv_prefix_list.append(cv_prefix)
for s in tqdm(X_test['teacher_prefix'].values):
    test_prefix = re.sub('[^A-Za-z0-9]+', '', str(s))
    test_prefix = re.sub('nan', '', str(test_prefix))
    X_test_prefix_list.append(test_prefix)

vectorizer = CountVectorizer(lowercase=False, binary=True)
X_train_one_hot_clean_teacher_prefix = vectorizer.fit_transform(X_train_prefix_list)
X_cv_one_hot_clean_teacher_prefix = vectorizer.transform(X_cv_prefix_list)
X_test_one_hot_clean_teacher_prefix = vectorizer.fit_transform(X_test_prefix_list)
print(vectorizer.get_feature_names())

print("Shape of matrix after one hot encodig ",X_train_one_hot_clean_teacher_prefix.shape)
print("Shape of matrix after one hot encodig ",X_cv_one_hot_clean_teacher_prefix.shape)
print("Shape of matrix after one hot encodig ",X_test_one_hot_clean_teacher_prefix.shape)
```

```
100%|██████████| 29178/29178 [00:00<00:00, 274311.04it/s]
100%|██████████| 14372/14372 [00:00<00:00, 424424.15it/s]
100%|██████████| 21450/21450 [00:00<00:00, 287845.44it/s]
['Dr', 'Mr', 'Mrs', 'Ms', 'Teacher']
```

```
with open("/content/drive/My Drive/appliedai/New_dumps/X_train_one_hot_clean_teacher_prefi
  pickle.dump(X_train_one_hot_clean_teacher_prefix, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_cv_one_hot_clean_teacher_prefix.p
  pickle.dump(X_cv_one_hot_clean_teacher_prefix, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_test_one_hot_clean_teacher_prefix
  pickle.dump(X_test_one_hot_clean_teacher_prefix, file)
```

Project Grade Category

```
# Project Grade Category

def grade_cat_cleaning(data):
  proj_grade_cat_list = []
  for grade in tqdm(data):
      grade_cat = re.sub('-',' to ', grade)
      grade_cat = re.sub('2',' two ', grade_cat)
      grade_cat = re.sub('3',' three ', grade_cat)
      grade_cat = re.sub('5',' five ', grade_cat)
      grade_cat = re.sub('6',' six ', grade_cat)
      grade_cat = re.sub('8',' eight ', grade_cat)
      grade_cat = re.sub('9',' nine ', grade_cat)
      grade_cat = re.sub('12',' twelve ', grade_cat)
      proj_grade_cat_list.append(grade_cat.lower().strip())
  return proj_grade_cat_list
X_train_proj_grade_cat = grade_cat_cleaning([sent for sent in X_train['project_grade_categ
X_cv_proj_grade_cat = grade_cat_cleaning([sent for sent in X_cv['project_grade_category'].
X_test_proj_grade_cat = grade_cat_cleaning([sent for sent in X_test['project_grade_categor
vectorizer = CountVectorizer(lowercase=False, binary=True)
X_train_one_hot_clean_project_grade = vectorizer.fit_transform(X_train_proj_grade_cat)
X_cv_one_hot_clean_project_grade = vectorizer.transform(X_cv_proj_grade_cat)
X_test_one_hot_clean_project_grade = vectorizer.transform(X_test_proj_grade_cat)
print(vectorizer.get_feature_names())

print("Shape of matrix after one hot encodig ",X_train_one_hot_clean_project_grade.shape)
print("Shape of matrix after one hot encodig ",X_cv_one_hot_clean_project_grade.shape)
print("Shape of matrix after one hot encodig ",X_test_one_hot_clean_project_grade.shape)
```

```
100%|██████████| 29178/29178 [00:00<00:00, 104215.76it/s]
100%|██████████| 14372/14372 [00:00<00:00, 104013.20it/s]
100%|██████████| 21450/21450 [00:00<00:00, 103407.96it/s]
['eight', 'five', 'grades', 'nine', 'prek', 'six', 'three', 'to', 'two']
Shape of matrix after one hot encodig  (29178, 9)
Shape of matrix after one hot encodig  (14372, 9)
Shape of matrix after one hot encodig  (21450, 9)
```

```
with open("/content/drive/My Drive/appliedai/New_dumps/X_train_one_hot_clean_project_grade
  pickle.dump(X_train_one_hot_clean_project_grade, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_cv_one_hot_clean_project_grade.pk
  pickle.dump(X_cv_one_hot_clean_project_grade, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_test_one_hot_clean_project_grade.
  pickle.dump(X_test_one_hot_clean_project_grade, file)
```

2.3 Make Data Model Ready: encoding eassay, and project_title

```
X_train.head(2)
```

id	cleaned_essay	cleaned_project_title	clean_categories	clean_subcategori
p104502	i class pretty awesome third grade students re...	headphones microscopes are what we need	Math_Science	AppliedScienc Mathemati
p231069	if saw kids i sure would amazed i students pub...	chromebook classroom	Literacy_Language	Literature_Writi

2.3.1: BOW

Cleaned Text

```
vectorizer = CountVectorizer(min_df=10)
X_train_bow_text = vectorizer.fit_transform(X_train['cleaned_text'])
X_cv_bow_text = vectorizer.transform(X_cv['cleaned_text'])
X_test_bow_text = vectorizer.transform(X_test['cleaned_text'])
print("Shape of matrix after BOW encodig ",X_train_bow_text.shape)
print("Shape of matrix after BOW encodig ",X_cv_bow_text.shape)
print("Shape of matrix after BOW encodig ",X_test_bow_text.shape)
```

```
Shape of matrix after BOW encodig  (29178, 10437)
Shape of matrix after BOW encodig  (14372, 10437)
Shape of matrix after BOW encodig  (21450, 10437)
```

```
with open("/content/drive/My Drive/appliedai/New_dumps/X_train_bow_text.pkl","wb") as file
  pickle.dump(X_train_bow_text, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_cv_bow_text.pkl","wb") as file:
  pickle.dump(X_cv_bow_text, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_test_bow_text.pkl","wb") as file:
  pickle.dump(X_test_bow_text, file)
```

2.3.2: TFIDF Vectorizer

Cleaned Text

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
X_train_tfidf_text = vectorizer.fit_transform(X_train['cleaned_text'])
X_cv_tfidf_text = vectorizer.transform(X_cv['cleaned_text'])
X_test_tfidf_text = vectorizer.transform(X_test['cleaned_text'])
print("Shape of matrix after TFIDF encodig ",X_train_tfidf_text.shape)
print("Shape of matrix after TFIDF encodig ",X_cv_tfidf_text.shape)
print("Shape of matrix after TFIDF encodig ",X_test_tfidf_text.shape)
```

```
Shape of matrix after TFIDF encodig  (29178, 10437)
Shape of matrix after TFIDF encodig  (14372, 10437)
Shape of matrix after TFIDF encodig  (21450, 10437)
```

```
with open("/content/drive/My Drive/appliedai/New_dumps/X_train_tfidf_text.pkl","wb") as fi
  pickle.dump(X_train_tfidf_text, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_cv_tfidf_text.pkl","wb") as file:
  pickle.dump(X_cv_tfidf_text, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_test_tfidf_text.pkl","wb") as fil
  pickle.dump(X_test_tfidf_text, file)
```

2.3.3: Avg W2V

Cleaned Text

```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
```

```
X_train_list_of_sent=[]
X_cv_list_of_sent=[]
X_test_list_of_sent=[]
for sent in X_train['cleaned_text'].values:
    X_train_list_of_sent.append(sent.split())
for sent in X_cv['cleaned_text'].values:
    X_cv_list_of_sent.append(sent.split())
for sent in X_test['cleaned_text'].values:
    X_test_list_of_sent.append(sent.split())
```

```
#train data avg w2v
w2v_model=Word2Vec(X_train_list_of_sent,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
```

```
# average Word2Vec
# compute average word2vec for each text.
def avg_w2v(sent_list):
  sent_vectors = []; # the avg-w2v for each sentence is stored in this list
  for sent in tqdm(sent_list): # for each tain sentence
      sent_vec = np.zeros(50) # as word vectors are of zero length
      cnt_words =0; # num of words with a valid vector in the sentence
      for word in sent: # for each word in a sentence
          if word in w2v_words:
              vec = w2v_model.wv[word]
              sent_vec += vec
              cnt_words += 1
      if cnt_words != 0:
          sent_vec /= cnt_words
      sent_vectors.append(sent_vec)
  print("\n",len(sent_vectors))
  print(len(sent_vectors[0]))
  return sent_vectors
```

```
X_train_avg_w2v_text = avg_w2v([sent.split() for sent in X_train['cleaned_text']])
X_cv_avg_w2v_text = avg_w2v([sent.split() for sent in X_cv['cleaned_text']])
X_test_avg_w2v_text = avg_w2v([sent.split() for sent in X_test['cleaned_text']])
```

```
print("Shape of matrix after Avg W2V encodig ",len(X_train_avg_w2v_text))
print("Shape of matrix after Avg W2V encodig ",len(X_cv_avg_w2v_text))
print("Shape of matrix after Avg W2V encodig ",len(X_test_avg_w2v_text))
```

```
100%|██████████| 29178/29178 [02:21<00:00, 206.68it/s]

 29178
50
100%|██████████| 14372/14372 [01:10<00:00, 202.59it/s]

 14372
50
100%|██████████| 21450/21450 [01:45<00:00, 202.82it/s]

 21450
50
Shape of matrix after Avg W2V encodig  29178
Shape of matrix after Avg W2V encodig  14372
Shape of matrix after Avg W2V encodig  21450
```

```
with open("/content/drive/My Drive/appliedai/New_dumps/X_train_avg_w2v_text.pkl","wb") as
  pickle.dump(X_train_avg_w2v_text, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_cv_avg_w2v_text.pkl","wb") as fil
  pickle.dump(X_cv_avg_w2v_text, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_test_avg_w2v_text.pkl","wb") as f
  pickle.dump(X_test_avg_w2v_text, file)
```

2.3.4: TFIDF Weighted W2V

Cleaned Text

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_model = TfidfVectorizer()
X_train_tfidf_w2v_model_text = tfidf_model.fit_transform(X_train['cleaned_text'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

```
# TF-IDF weighted Word2Vec
tfidf_features = tfidf_model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

def tfidf_w2v(sen_list):
  tfidf_w2v_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
  row=0;

  for sent in tqdm(sen_list): # for each review/sentence
      vector = np.zeros(50) # as word vectors are of zero length
      tf_idf_weight =0; # num of words with a valid vector in the sentence/review
      for word in sent: # for each word in a review/sentence
          if (word in w2v_words) and (word in tfidf_features):
              vec = w2v_model.wv[word]
              tf_idf = dictionary[word]*(sent.count(word)/len(sent))
              vector += (vec * tf_idf)
              tf_idf_weight += tf_idf
      if tf_idf_weight != 0:
          vector /= tf_idf_weight
      tfidf_w2v_vectors.append(vector)
      row += 1
  return tfidf_w2v_vectors


X_train_tfidf_w2v_text = tfidf_w2v([sent.split() for sent in X_train['cleaned_text']])
X_cv_tfidf_w2v_text = tfidf_w2v([sent.split() for sent in X_cv['cleaned_text']])
X_test_tfidf_w2v_text = tfidf_w2v([sent.split() for sent in X_test['cleaned_text']])
print("Shape of matrix after Avg W2V encodig ",len(X_train_tfidf_w2v_text))
print("Shape of matrix after Avg W2V encodig ",len(X_cv_tfidf_w2v_text))
print("Shape of matrix after Avg W2V encodig ",len(X_test_tfidf_w2v_text))
```

```
100%|██████████| 29178/29178 [39:58<00:00, 11.52it/s]
100%|██████████| 14372/14372 [19:41<00:00, 11.23it/s]
100%|██████████| 21450/21450 [29:38<00:00, 10.58it/s]Shape of matrix after Avg W2V
Shape of matrix after Avg W2V encodig  14372
Shape of matrix after Avg W2V encodig  21450
```

```
with open("/content/drive/My Drive/appliedai/New_dumps/X_train_tfidf_w2v_text.pkl","wb") a
  pickle.dump(X_train_tfidf_w2v_text, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_cv_tfidf_w2v_text.pkl","wb") as f
  pickle.dump(X_cv_tfidf_w2v_text, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_test_tfidf_w2v_text.pkl","wb") as
  pickle.dump(X_test_tfidf_w2v_text, file)
```

Merging Categorical and Numerical Features

```
from scipy.sparse import csr_matrix
X_train_no_of_projects = np.array(X_train['teacher_number_of_previously_posted_projects'])
X_train_no_of_projects = csr_matrix(X_train_no_of_projects).T
X_cv_no_of_projects = np.array(X_cv['teacher_number_of_previously_posted_projects'])
X_cv_no_of_projects = csr_matrix(X_cv_no_of_projects).T
X_test_no_of_projects = np.array(X_test['teacher_number_of_previously_posted_projects'])
X_test_no_of_projects = csr_matrix(X_test_no_of_projects).T
print(X_train_no_of_projects.shape)
print(X_cv_no_of_projects.shape)
print(X_test_no_of_projects.shape)
```

```
(29178, 1)
(14372, 1)
(21450, 1)
```

```
from scipy.sparse import coo_matrix, hstack
print(X_train_one_hot_clean_cat.shape, X_train_one_hot_clean_sub_cat.shape, X_train_one_ho
                                 X_train_one_hot_clean_teacher_prefix.shape, X_train_one_h
X_train_categorical_numerical = hstack([X_train_one_hot_clean_cat, X_train_one_hot_clean_s
                              X_train_one_hot_clean_teacher_prefix, X_train_one_hot_clean_
X_cv_categorical_numerical = hstack([X_cv_one_hot_clean_cat, X_cv_one_hot_clean_sub_cat, X
                                X_cv_one_hot_clean_teacher_prefix, X_cv_one_hot_clean_proj
X_test_categorical_numerical = hstack([X_test_one_hot_clean_cat, X_test_one_hot_clean_sub_
                                X_test_one_hot_clean_teacher_prefix, X_test_one_hot_clean_
print(X_train_categorical_numerical.shape, X_cv_categorical_numerical.shape, X_test_catego
```

```
(29178, 9) (29178, 30) (29178, 51) (29178, 5) (29178, 9)
(29178, 105) (14372, 105) (21450, 105)
```

SET 1: Merging: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)

```
X_train_bow_feat = hstack([X_train_categorical_numerical, X_train_bow_text])
X_cv_bow_feat = hstack([X_cv_categorical_numerical, X_cv_bow_text])
X_test_bow_feat = hstack([X_test_categorical_numerical, X_test_bow_text])
print(X_train_bow_feat.shape)
print(X_cv_bow_feat.shape)
print(X_test_bow_feat.shape)
```

```
(29178, 10542)
(14372, 10542)
(21450, 10542)
```

```
with open("/content/drive/My Drive/appliedai/New_dumps/X_train_bow_feat.pkl","wb") as file
  pickle.dump(X_train_bow_feat, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_cv_bow_feat.pkl","wb") as file:
  pickle.dump(X_cv_bow_feat, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_test_bow_feat.pkl","wb") as file:
  pickle.dump(X_test_bow_feat, file)
```

SET 2: Merging: categorical, numerical features + project_title(TFIDF) + preprocessed_essay (TFIDF)

```
X_train_tfidf_feat = hstack([X_train_categorical_numerical, X_train_tfidf_text])
X_cv_tfidf_feat = hstack([X_cv_categorical_numerical, X_cv_tfidf_text])
X_test_tfidf_feat = hstack([X_test_categorical_numerical, X_test_tfidf_text])
print(X_train_tfidf_feat.shape)
print(X_cv_tfidf_feat.shape)
print(X_test_tfidf_feat.shape)
```

```
(29178, 10542)
(14372, 10542)
(21450, 10542)
```

```
with open("/content/drive/My Drive/appliedai/New_dumps/X_train_tfidf_feat.pkl","wb") as fi
  pickle.dump(X_train_tfidf_feat, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_cv_tfidf_feat.pkl","wb") as file:
  pickle.dump(X_cv_tfidf_feat, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_test_tfidf_feat.pkl","wb") as fil
  pickle.dump(X_test_tfidf_feat, file)
```

SET 3: Merging: categorical, numerical features + project_title(Avg W2V) + preprocessed_essay (Avg W2V)

```
X_train_avg_w2v_feat = hstack([X_train_categorical_numerical, X_train_avg_w2v_text])
X_cv_avg_w2v_feat = hstack([X_cv_categorical_numerical, X_cv_avg_w2v_text])
X_test_avg_w2v_feat = hstack([X_test_categorical_numerical, X_test_avg_w2v_text])
print(X_train_avg_w2v_feat.shape)
print(X_cv_avg_w2v_feat.shape)
print(X_test_avg_w2v_feat.shape)
```

```
(29178, 155)
(14372, 155)
(21450, 155)
```

```
with open("/content/drive/My Drive/appliedai/New_dumps/X_train_avg_w2v_feat.pkl","wb") as
  pickle.dump(X_train_avg_w2v_feat, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_cv_avg_w2v_feat.pkl","wb") as fil
  pickle.dump(X_cv_avg_w2v_feat, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_test_avg_w2v_feat.pkl","wb") as f
  pickle.dump(X_test_avg_w2v_feat, file)
```

SET 4: Merging: categorical, numerical features + project_title(TFIDF W2V) + preprocessed_essay (TFIDF W2V)

```
X_train_tfidf_w2v_feat = hstack([X_train_categorical_numerical, X_train_tfidf_w2v_text])
X_cv_tfidf_w2v_feat = hstack([X_cv_categorical_numerical, X_cv_tfidf_w2v_text])
X_test_tfidf_w2v_feat = hstack([X_test_categorical_numerical, X_test_tfidf_w2v_text])
print(X_train_tfidf_w2v_feat.shape)
print(X_cv_tfidf_w2v_feat.shape)
print(X_test_tfidf_w2v_feat.shape)
```

```
(29178, 155)
(14372, 155)
(21450, 155)
```

```
with open("/content/drive/My Drive/appliedai/New_dumps/X_train_tfidf_w2v_feat.pkl","wb") a
  pickle.dump(X_train_tfidf_w2v_feat, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_cv_tfidf_w2v_feat.pkl","wb") as f
  pickle.dump(X_cv_tfidf_w2v_feat, file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_test_tfidf_w2v_feat.pkl","wb") as
  pickle.dump(X_test_tfidf_w2v_feat, file)
```

2.4 Appling KNN on different kind of featurization as mentioned in the instructions

Apply KNN on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instructions

2.4.1 Applying KNN brute force on BOW, SET 1

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
```

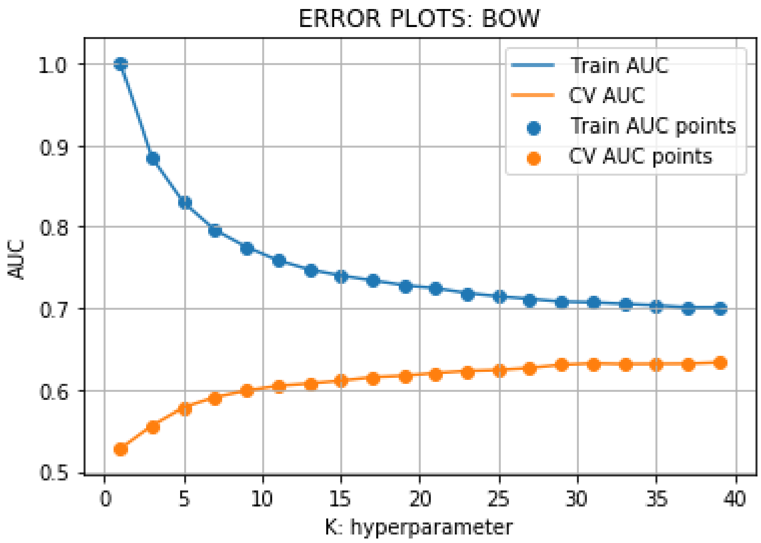
```
list(range(1, 40, 2))
```

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39]
```

```
# Creating odd list of K for KNN
neighbors = range(1, 40, 2)
print(neighbors)
train_auc = []
cv_auc = []
for k in tqdm(neighbors):
    neigh = KNeighborsClassifier(n_neighbors=k, algorithm='brute')
    neigh.fit(X_train_bow_feat, y_train)

    #predict probabilities for train and validation
    y_train_pred = neigh.predict_proba(X_train_bow_feat)[:,1]
    y_cv_pred = neigh.predict_proba(X_cv_bow_feat)[:,1]

#     y_train_pred = batch_predict(neigh, X_tr)
#     y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
  0%|          | 0/20 [00:00<?, ?it/s]range(1, 40, 2)
100%|██████████| 20/20 [1:01:20<00:00, 185.11s/it]
```

```
plt.plot(neighbors, train_auc, label='Train AUC')
plt.plot(neighbors, cv_auc, label='CV AUC')

plt.scatter(neighbors, train_auc, label='Train AUC points')
plt.scatter(neighbors, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS: BOW")
plt.grid()
plt.show()
```



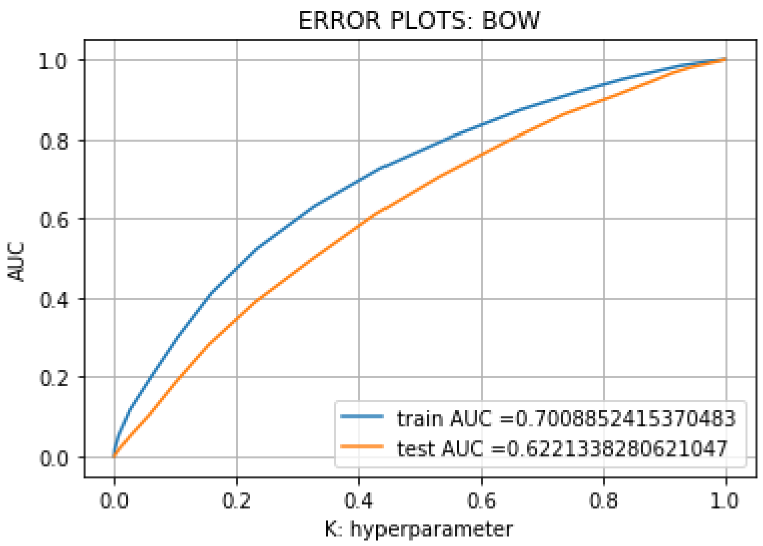
```
best_k = neighbors[cv_auc.index(max(cv_auc))]
print(best_k)
print(max(cv_auc))
```

```
39
0.6333217924144336
```

Observation: From the above graph we got the best K value at K=37

```
# Training KNN with best K
from sklearn.metrics import roc_curve, auc, classification_report
best_k = neighbors[cv_auc.index(max(cv_auc))]
neigh = KNeighborsClassifier(n_neighbors=best_k, algorithm='brute')
neigh.fit(X_train_bow_feat, y_train)
```

```
#predict probabilities for train and test
y_train_pred = neigh.predict_proba(X_train_bow_feat)[:,1]
y_test_pred = neigh.predict_proba(X_test_bow_feat)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)


plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS: BOW")
plt.grid()
plt.show()
```



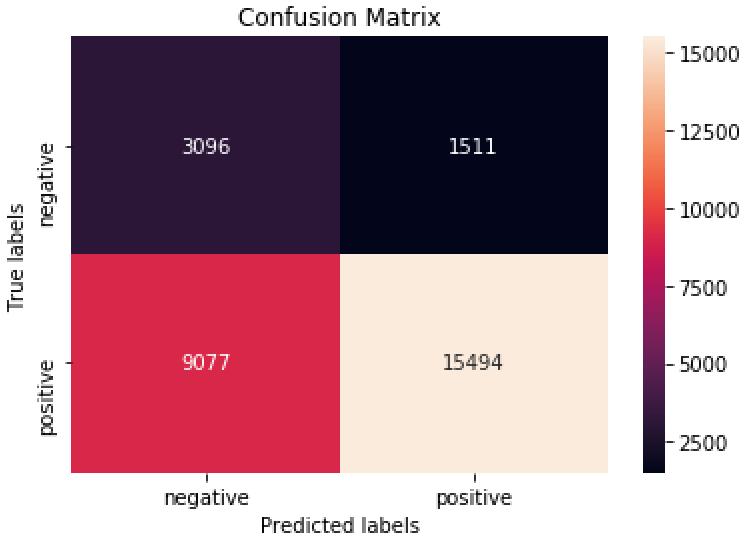
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

```
print("="*100)
from sklearn.metrics import confusion_matrix, classification_report
print("Train confusion matrix")
conf_matrix = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, tr
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
```

```
==========================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.4237634146649361 for threshold 0.795
the maximum value of tpr*(1-fpr) 0.4237634146649361 for threshold 0.795
[[ 3096  1511]
```

```
import seaborn as sn
import matplotlib.pyplot as plt
heat_map = plt.subplot()
sn.heatmap(conf_matrix, annot=True, ax = heat_map, fmt='g')

heat_map.set_ylabel('True labels')
heat_map.set_xlabel('Predicted labels')
heat_map.set_title('Confusion Matrix')
heat_map.xaxis.set_ticklabels(['negative', 'positive'])
heat_map.yaxis.set_ticklabels(['negative', 'positive'])
```

```
[Text(0, 0.5, 'negative'), Text(0, 1.5, 'positive')]
```

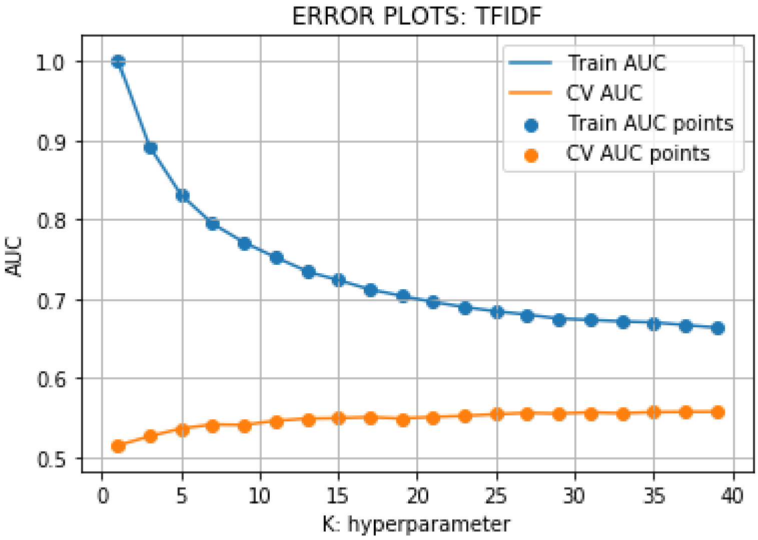


2.4.2 Applying KNN brute force on TFIDF, SET 2

```
# Creating odd list of K for KNN
neighbors = range(1, 40, 2)
print(neighbors)
train_auc = []
cv_auc = []
for k in tqdm(neighbors):
    neigh = KNeighborsClassifier(n_neighbors=k, algorithm='brute')
    neigh.fit(X_train_tfidf_feat, y_train)

    #predict probabilities for train and validation
    y_train_pred = neigh.predict_proba(X_train_tfidf_feat)[:,1]
    y_cv_pred = neigh.predict_proba(X_cv_tfidf_feat)[:,1]

#     y_train_pred = batch_predict(neigh, X_tr)
#     y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
  0%|          | 0/20 [00:00<?, ?it/s]range(1, 40, 2)
100%|██████████| 20/20 [1:01:09<00:00, 184.33s/it]
```

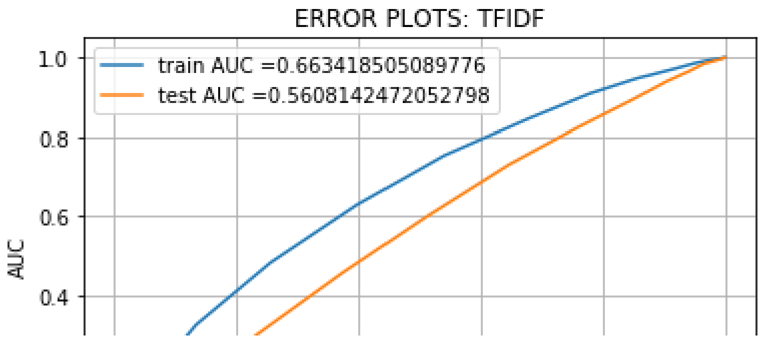
```
plt.plot(neighbors, train_auc, label='Train AUC')
plt.plot(neighbors, cv_auc, label='CV AUC')

plt.scatter(neighbors, train_auc, label='Train AUC points')
plt.scatter(neighbors, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS: TFIDF")
plt.grid()
plt.show()
```



```
best_k = neighbors[cv_auc.index(max(cv_auc))]
print(best_k)
print(max(cv_auc))
```

```
39
0.5571770902660931
```

```
# Training KNN with best K
from sklearn.metrics import roc_curve, auc, classification_report
best_k = neighbors[cv_auc.index(max(cv_auc))]
neigh = KNeighborsClassifier(n_neighbors=best_k, algorithm='brute')
neigh.fit(X_train_tfidf_feat, y_train)

#predict probabilities for train and test
y_train_pred = neigh.predict_proba(X_train_tfidf_feat)[:,1]
y_test_pred = neigh.predict_proba(X_test_tfidf_feat)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)


plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS: TFIDF")
plt.grid()
plt.show()
```

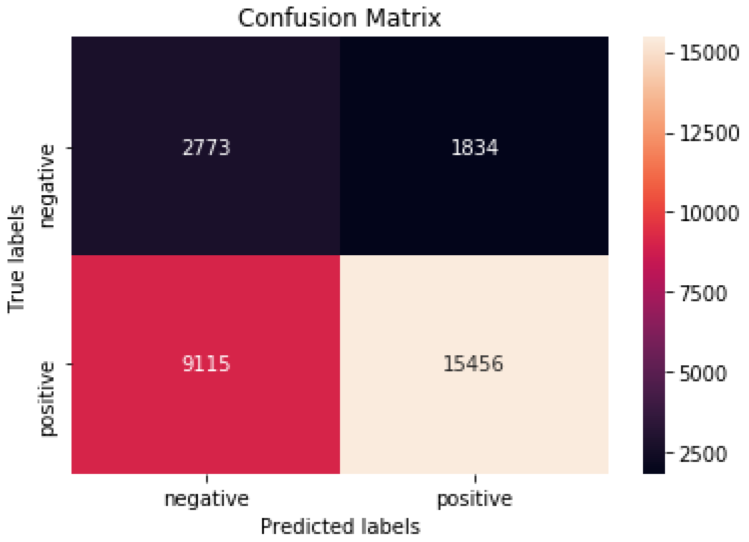


```
print("="*100)
from sklearn.metrics import confusion_matrix, classification_report
print("Train confusion matrix")
conf_matrix = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, tr
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
```

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.37862207779836704 for threshold 0.846
the maximum value of tpr*(1-fpr) 0.37862207779836704 for threshold 0.846
[[ 2773  1834]
 [ 9115 15456]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2922741241857285 for threshold 0.846
[[ 1645  1742]
 [ 7193 10870]]
```

```
import seaborn as sn
import matplotlib.pyplot as plt
heat_map = plt.subplot()
sn.heatmap(conf_matrix, annot=True, ax = heat_map, fmt='g')

heat_map.set_ylabel('True labels')
heat_map.set_xlabel('Predicted labels')
heat_map.set_title('Confusion Matrix')
heat_map.xaxis.set_ticklabels(['negative', 'positive'])
heat_map.yaxis.set_ticklabels(['negative', 'positive'])
```

```
[Text(0, 0.5, 'negative'), Text(0, 1.5, 'positive')]
```



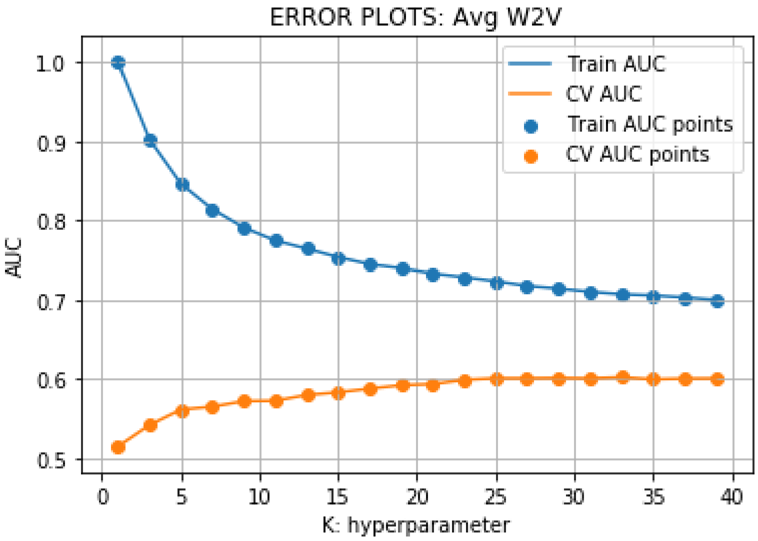
2.4.3 Applying KNN brute force on AVG W2V, SET 3

```
with open("/content/drive/My Drive/appliedai/New_dumps/X_train_avg_w2v_feat.pkl","rb") as
  X_train_avg_w2v_feat = pickle.load(file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_cv_avg_w2v_feat.pkl","rb") as fil
  X_cv_avg_w2v_feat = pickle.load(file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_test_avg_w2v_feat.pkl","rb") as f
  X_test_avg_w2v_feat = pickle.load(file)

with open("/content/drive/My Drive/appliedai/New_dumps/y_train.pkl","rb") as file:
  y_train = pickle.load(file)

with open("/content/drive/My Drive/appliedai/New_dumps/y_cv.pkl","rb") as file:
  y_cv = pickle.load(file)

with open("/content/drive/My Drive/appliedai/New_dumps/y_test.pkl","rb") as file:
  y_test = pickle.load(file)
```

```
# Creating odd list of K for KNN
neighbors = range(1, 40, 2)
print(neighbors)
train_auc = []
cv_auc = []
for k in tqdm(neighbors):
    neigh = KNeighborsClassifier(n_neighbors=k, algorithm='brute')
    neigh.fit(X_train_avg_w2v_feat, y_train)

    #predict probabilities for train and validation
    y_train_pred = neigh.predict_proba(X_train_avg_w2v_feat)[:,1]
    y_cv_pred = neigh.predict_proba(X_cv_avg_w2v_feat)[:,1]

#     y_train_pred = batch_predict(neigh, X_tr)
#     y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
  0%|          | 0/20 [00:00<?, ?it/s]range(1, 40, 2)

  5%|▌         | 1/20 [05:36<1:46:38, 336.77s/it]
 10%|█         | 2/20 [11:05<1:40:21, 334.50s/it]
 15%|█▌        | 3/20 [16:42<1:34:54, 334.98s/it]
 20%|██        | 4/20 [22:09<1:28:44, 332.77s/it]
 25%|██▌       | 5/20 [27:42<1:23:12, 332.84s/it]
 30%|███       | 6/20 [33:42<1:19:33, 340.96s/it]
 35%|███▌      | 7/20 [39:49<1:15:34, 348.80s/it]
 40%|████      | 8/20 [45:57<1:10:52, 354.40s/it]
 45%|████▌     | 9/20 [51:59<1:05:25, 356.83s/it]
 50%|█████     | 10/20 [57:52<59:17, 355.70s/it]
 55%|█████▌    | 11/20 [1:03:31<52:34, 350.55s/it]
 60%|██████    | 12/20 [1:09:15<46:29, 348.68s/it]
 65%|██████▌   | 13/20 [1:15:05<40:42, 348.96s/it]
 70%|███████   | 14/20 [1:20:53<34:51, 348.61s/it]
 75%|███████▌  | 15/20 [1:26:49<29:14, 350.87s/it]
 80%|████████  | 16/20 [1:32:47<23:32, 353.02s/it]
 85%|████████▌ | 17/20 [1:38:46<17:44, 354.84s/it]
 90%|█████████ | 18/20 [1:44:44<11:51, 356.00s/it]
 95%|█████████▌| 19/20 [1:50:45<05:57, 357.29s/it]
100%|██████████| 20/20 [1:56:45<00:00, 358.19s/it]
```

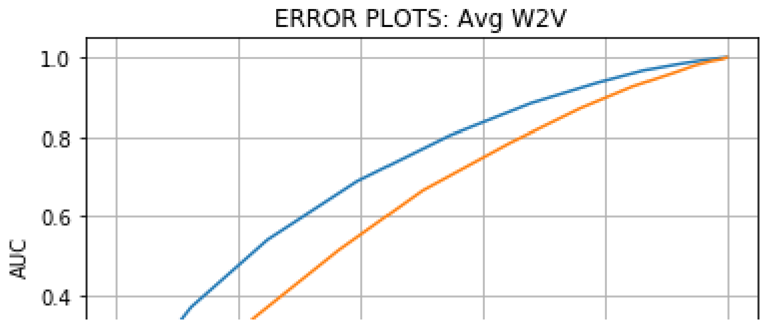
```
plt.plot(neighbors, train_auc, label='Train AUC')
plt.plot(neighbors, cv_auc, label='CV AUC')
plt.scatter(neighbors, train_auc, label='Train AUC points')
plt.scatter(neighbors, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS: Avg W2V")
plt.grid()
plt.show()
```



```
best_k = neighbors[cv_auc.index(max(cv_auc))]
print(best_k)
print(max(cv_auc))
```

```
33
0.6019682607494137
```

```
# Training KNN with best K
from sklearn.metrics import roc_curve, auc, classification_report
best_k = neighbors[cv_auc.index(max(cv_auc))]
neigh = KNeighborsClassifier(n_neighbors=best_k, algorithm='brute')
neigh.fit(X_train_avg_w2v_feat, y_train)

#predict probabilities for train and test
y_train_pred = neigh.predict_proba(X_train_avg_w2v_feat)[:,1]
y_test_pred = neigh.predict_proba(X_test_avg_w2v_feat)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)


plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS: Avg W2V")
plt.grid()
plt.show()
```

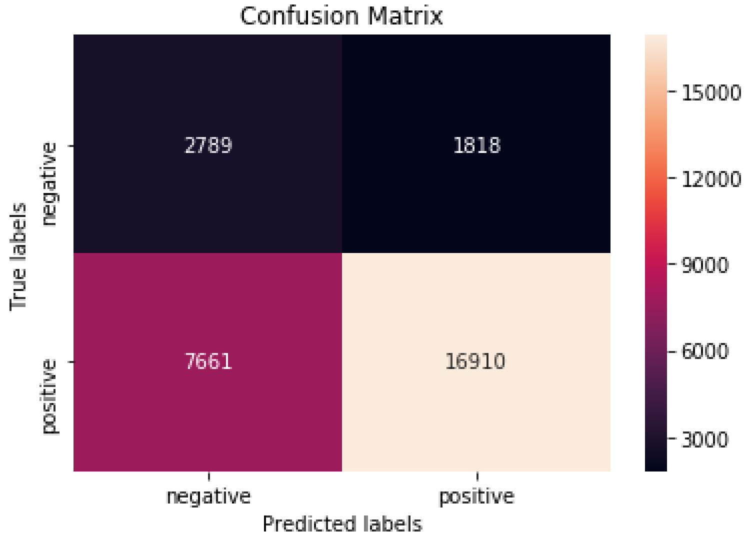


```
print("="*100)
from sklearn.metrics import confusion_matrix, classification_report
print("Train confusion matrix")
conf_matrix = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, tr
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
```

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.41663051707257465 for threshold 0.848
the maximum value of tpr*(1-fpr) 0.41663051707257465 for threshold 0.848
[[ 2789  1818]
 [ 7661 16910]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.3312266922086054 for threshold 0.848
[[ 1687  1700]
 [ 6051 12012]]
```

```
import seaborn as sn
import matplotlib.pyplot as plt
heat_map = plt.subplot()
sn.heatmap(conf_matrix, annot=True, ax = heat_map, fmt='g')

heat_map.set_ylabel('True labels')
heat_map.set_xlabel('Predicted labels')
heat_map.set_title('Confusion Matrix')
heat_map.xaxis.set_ticklabels(['negative', 'positive'])
heat_map.yaxis.set_ticklabels(['negative', 'positive'])
```

```
[Text(0, 0.5, 'negative'), Text(0, 1.5, 'positive')]
```



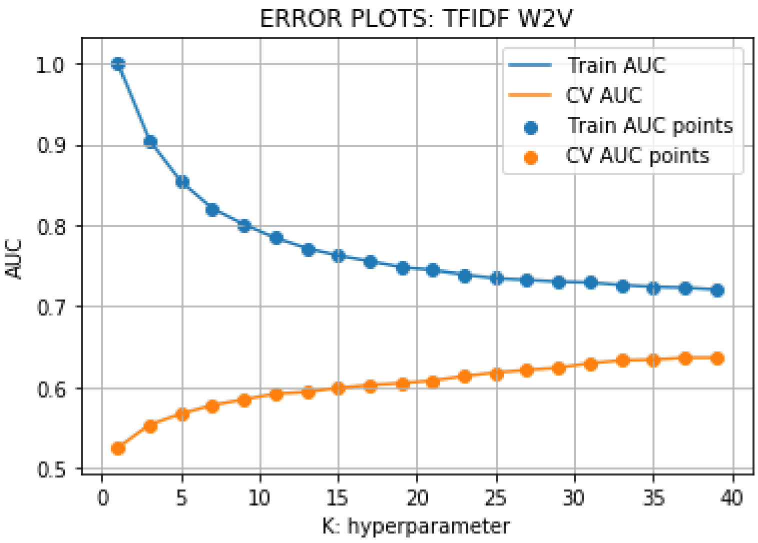
2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

```
with open("/content/drive/My Drive/appliedai/New_dumps/X_train_tfidf_w2v_feat.pkl","rb") a
  X_train_tfidf_w2v_feat = pickle.load(file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_cv_tfidf_w2v_feat.pkl","rb") as f
  X_cv_tfidf_w2v_feat = pickle.load(file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_test_tfidf_w2v_feat.pkl","rb") as
  X_test_tfidf_w2v_feat = pickle.load(file)

with open("/content/drive/My Drive/appliedai/New_dumps/y_train.pkl","rb") as file:
  y_train = pickle.load(file)

with open("/content/drive/My Drive/appliedai/New_dumps/y_cv.pkl","rb") as file:
  y_cv = pickle.load(file)

with open("/content/drive/My Drive/appliedai/New_dumps/y_test.pkl","rb") as file:
  y_test = pickle.load(file)
```

```
# Creating odd list of K for KNN
neighbors = range(1, 40, 2)
print(neighbors)
train_auc = []
cv_auc = []
for k in tqdm(neighbors):
    neigh = KNeighborsClassifier(n_neighbors=k, algorithm='brute')
    neigh.fit(X_train_tfidf_w2v_feat, y_train)

    #predict probabilities for train and validation
    y_train_pred = neigh.predict_proba(X_train_tfidf_w2v_feat)[:,1]
    y_cv_pred = neigh.predict_proba(X_cv_tfidf_w2v_feat)[:,1]

#     y_train_pred = batch_predict(neigh, X_tr)
#     y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
  0%|          | 0/20 [00:00<?, ?it/s]range(1, 40, 2)

  5%|▌         | 1/20 [05:51<1:51:16, 351.38s/it]
 10%|█         | 2/20 [11:46<1:45:44, 352.45s/it]
 15%|█▌        | 3/20 [17:46<1:40:31, 354.82s/it]
 20%|██        | 4/20 [23:49<1:35:15, 357.23s/it]
 25%|██▌       | 5/20 [29:53<1:29:49, 359.27s/it]
 30%|███       | 6/20 [36:00<1:24:20, 361.47s/it]
 35%|███▌      | 7/20 [42:06<1:18:37, 362.88s/it]
 40%|████      | 8/20 [48:09<1:12:36, 363.02s/it]
 45%|████▌     | 9/20 [54:10<1:06:27, 362.50s/it]
 50%|█████     | 10/20 [1:00:08<1:00:09, 360.95s/it]
 55%|█████▌    | 11/20 [1:06:04<53:55, 359.55s/it]
 60%|██████    | 12/20 [1:12:03<47:55, 359.46s/it]
 65%|██████▌   | 13/20 [1:18:04<41:57, 359.69s/it]
 70%|███████   | 14/20 [1:24:01<35:53, 358.99s/it]
 75%|███████▌  | 15/20 [1:30:01<29:57, 359.42s/it]
 80%|████████  | 16/20 [1:36:03<23:59, 359.94s/it]
 85%|████████▌ | 17/20 [1:42:02<17:59, 359.77s/it]
 90%|█████████ | 18/20 [1:48:03<12:00, 360.09s/it]
 95%|█████████▌| 19/20 [1:54:07<06:01, 361.26s/it]
100%|██████████| 20/20 [2:00:10<00:00, 361.89s/it]
```

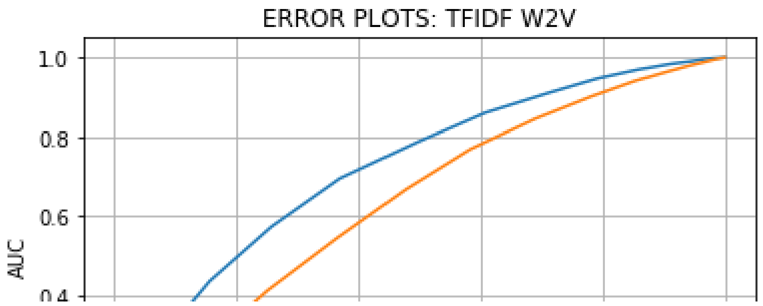
```
plt.plot(neighbors, train_auc, label='Train AUC')
plt.plot(neighbors, cv_auc, label='CV AUC')
plt.scatter(neighbors, train_auc, label='Train AUC points')
plt.scatter(neighbors, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS: TFIDF W2V")
plt.grid()
plt.show()
```



```
best_k = neighbors[cv_auc.index(max(cv_auc))]
print(best_k)
print(max(cv_auc))
```

```
39
0.6363421982471812
```

```
# Training KNN with best K
from sklearn.metrics import roc_curve, auc, classification_report
best_k = neighbors[cv_auc.index(max(cv_auc))]
neigh = KNeighborsClassifier(n_neighbors=best_k, algorithm='brute')
neigh.fit(X_train_tfidf_w2v_feat, y_train)

#predict probabilities for train and test
y_train_pred = neigh.predict_proba(X_train_tfidf_w2v_feat)[:,1]
y_test_pred = neigh.predict_proba(X_test_tfidf_w2v_feat)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)


plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS: TFIDF W2V")
plt.grid()
plt.show()
```

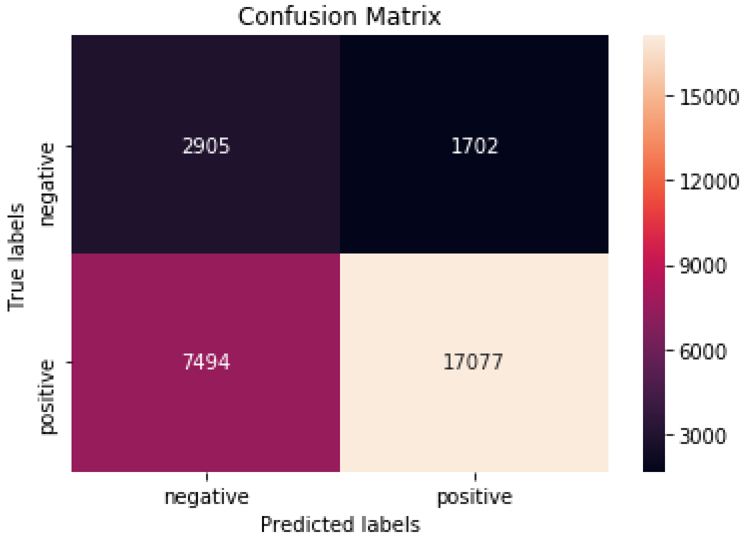


```
print("="*100)
from sklearn.metrics import confusion_matrix, classification_report
print("Train confusion matrix")
conf_matrix = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, tr
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
```

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.4382446983861469 for threshold 0.846
the maximum value of tpr*(1-fpr) 0.4382446983861469 for threshold 0.846
[[ 2905  1702]
 [ 7494 17077]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.34874345982676747 for threshold 0.846
[[ 1769  1618]
 [ 6002 12061]]
```

```
import seaborn as sn
import matplotlib.pyplot as plt
heat_map = plt.subplot()
sn.heatmap(conf_matrix, annot=True, ax = heat_map, fmt='g')

heat_map.set_ylabel('True labels')
heat_map.set_xlabel('Predicted labels')
heat_map.set_title('Confusion Matrix')
heat_map.xaxis.set_ticklabels(['negative', 'positive'])
heat_map.yaxis.set_ticklabels(['negative', 'positive'])
```

```
[Text(0, 0.5, 'negative'), Text(0, 1.5, 'positive')]
```



2.5 Feature selection with `SelectKBest`

```
with open("/content/drive/My Drive/appliedai/New_dumps/X_train_tfidf_feat.pkl","rb") as fi
  X_train_tfidf_feat = pickle.load(file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_cv_tfidf_feat.pkl","rb") as file:
  X_cv_tfidf_feat = pickle.load(file)

with open("/content/drive/My Drive/appliedai/New_dumps/X_test_tfidf_feat.pkl","rb") as fil
  X_test_tfidf_feat = pickle.load(file)

with open("/content/drive/My Drive/appliedai/New_dumps/y_train.pkl","rb") as file:
  y_train = pickle.load(file)
```

```
from sklearn.feature_selection import SelectKBest, chi2
```

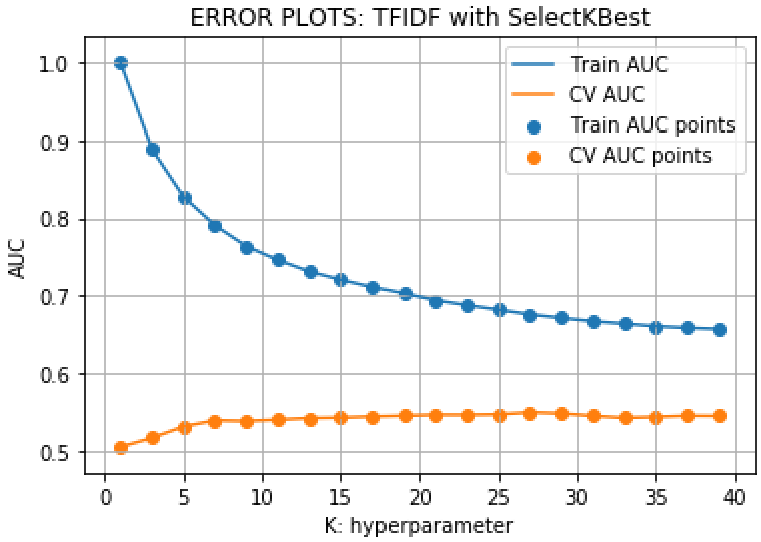
```
print(X_train_tfidf_feat.shape)
print(y_train.shape)
best_select_k = SelectKBest(chi2, k=2000)#.fit_transform(X_train_tfidf_feat, y_train)
X_train_new = best_select_k.fit_transform(X_train_tfidf_feat, y_train)
X_cv_new = best_select_k.transform(X_cv_tfidf_feat)
X_test_new = best_select_k.transform(X_test_tfidf_feat)
print(X_train_new.shape)
print(X_cv_new.shape)
print(X_test_new.shape)
```

```
(29178, 10542)
(29178,)
(29178, 2000)
(14372, 2000)
(21450, 2000)
```

```
# Creating odd list of K for KNN
neighbors = range(1, 40, 2)
print(neighbors)
train_auc = []
cv_auc = []
for k in tqdm(neighbors):
    neigh = KNeighborsClassifier(n_neighbors=k, algorithm='brute')
    neigh.fit(X_new, y_train)

    #predict probabilities for train and validation
    y_train_pred = neigh.predict_proba(X_new)[:,1]
    y_cv_pred = neigh.predict_proba(X_cv_new)[:,1]

#     y_train_pred = batch_predict(neigh, X_tr)
#     y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
  0%|          | 0/20 [00:00<?, ?it/s]range(1, 40, 2)

  5%|▌         | 1/20 [01:36<30:40, 96.86s/it]
 10%|█         | 2/20 [03:14<29:08, 97.14s/it]
 15%|█▌        | 3/20 [04:57<27:59, 98.78s/it]
 20%|██        | 4/20 [06:41<26:48, 100.51s/it]
 25%|██▌       | 5/20 [08:26<25:27, 101.85s/it]
 30%|███       | 6/20 [10:13<24:08, 103.45s/it]
 35%|███▌      | 7/20 [12:00<22:35, 104.25s/it]
 40%|████      | 8/20 [13:44<20:52, 104.42s/it]
 45%|████▌     | 9/20 [15:28<19:07, 104.28s/it]
 50%|█████     | 10/20 [17:13<17:24, 104.41s/it]
 55%|█████▌    | 11/20 [18:57<15:39, 104.35s/it]
 60%|██████    | 12/20 [20:42<13:55, 104.49s/it]
 65%|██████▌   | 13/20 [22:25<12:09, 104.15s/it]
```

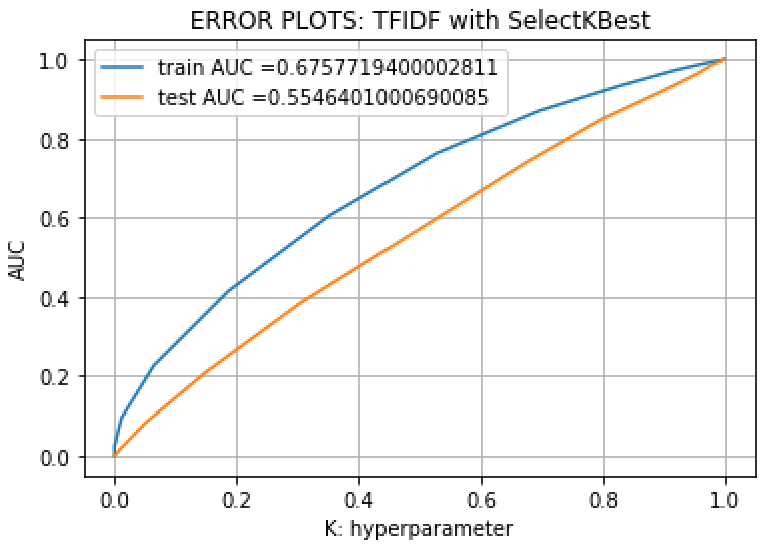
```python
plt.plot(neighbors, train_auc, label='Train AUC')
plt.plot(neighbors, cv_auc, label='CV AUC')

plt.scatter(neighbors, train_auc, label='Train AUC points')
plt.scatter(neighbors, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS: TFIDF with SelectKBest")
plt.grid()
plt.show()
```



```
best_k = neighbors[cv_auc.index(max(cv_auc))]
print(best_k)
print(max(cv_auc))
```

```
27
0.549245245388424
```

```
# Training KNN with best K
from sklearn.metrics import roc_curve, auc, classification_report
best_k = neighbors[cv_auc.index(max(cv_auc))]
neigh = KNeighborsClassifier(n_neighbors=best_k, algorithm='brute')
neigh.fit(X_new, y_train)

#predict probabilities for train and test
y_train_pred = neigh.predict_proba(X_new)[:,1]
y_test_pred = neigh.predict_proba(X_test_new)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)


plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS: TFIDF with SelectKBest")
plt.grid()
plt.show()
```

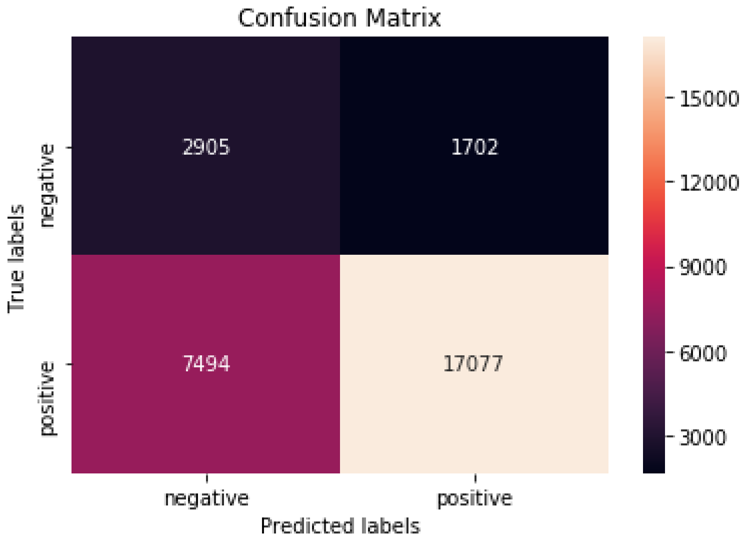


```
print("="*100)
from sklearn.metrics import confusion_matrix, classification_report
print("Train confusion matrix")
conf_matrix = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, tr
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
```

```
--------------------------------------------------------------------------------
```

```
import seaborn as sn
import matplotlib.pyplot as plt
heat_map = plt.subplot()
sn.heatmap(conf_matrix, annot=True, ax = heat_map, fmt='g')

heat_map.set_ylabel('True labels')
heat_map.set_xlabel('Predicted labels')
heat_map.set_title('Confusion Matrix')
heat_map.xaxis.set_ticklabels(['negative', 'positive'])
heat_map.yaxis.set_ticklabels(['negative', 'positive'])
```

```
[Text(0, 0.5, 'negative'), Text(0, 1.5, 'positive')]
```



3. Conclusions

```
from prettytable import PrettyTable


knn = PrettyTable()
knn.field_names = ["Vectorization Method", "Algorithm", "Hyperparameter(K)", "AUC"]


knn.add_row(["BOW", "Brute Force", 39, 0.6333])
knn.add_row(["TFIDF", "Brute Force", 39, 0.5571])
knn.add_row(["Avg W2V", "Brute Force", 33, 0.6019])
knn.add_row(["TFIDF Weighted W2V", "Brute Force", 39, 0.6363])
knn.add_row(["TFIDF: SelectKBest", "Brute Force", 27, 0.5492])
print(knn)
```

```
+----------------------+-------------+-------------------+--------+
| Vectorization Method |  Algorithm  | Hyperparameter(K) |  AUC   |
+----------------------+-------------+-------------------+--------+
|         BOW          | Brute Force |         39        | 0.6333 |
|        TFIDF         | Brute Force |         39        | 0.5571 |
|       Avg W2V        | Brute Force |         33        | 0.6019 |
|  TFIDF Weighted W2V  | Brute Force |         39        | 0.6363 |
|  TFIDF: SelectKBest  | Brute Force |         27        | 0.5492 |
+----------------------+-------------+-------------------+--------+
```